Lesson 2 - Vectorization

From what we learned so far, there isn't much we can do with MATLAB right now. But, MATLAB has introduced some nifty features that make calculations easier for us. Vectorization in MATLAB helps improve performance over traditional programming structures like loops and scalars as MATLAB is optimized for vectors and matrices.

Making Vectors and Matrices

There are some very common vectors that we would like to create easily in MATLAB. So, MATLAB has coded some nice shortcuts to make sure that we can.

```matlab
asc = 0:8;
% Should output [0, 1, 2, 3, 4, 5, 6, 7, 8]
morespaced = 0:2:16;
desc = 5:-1:0;
% You can adjust the step size with a third argument. The variables above should output [0, 2, 4, 6, 8, 10, 12, 14, 16] and [5, 4, 3, 2, 1, 0]

spacing = linspace(0, pi, 1000);
% This gives you 1000 equally spaced points from 0 to pi (both included)

complexspace = linspace(1i, 10-5i, 10);
% Linspace also works for complex numbers
```

Typing in all of the matrix elements one by one in the command line is very tedious, so MATLAB has some other fast ways to create other matrices that we want.

```matlab
allzeros = zeros(3, 1);
% Gives a 3x1 matrix with all zeros.

allones = ones(5, 2);
% Gives a 5x2 matrix with all ones.

identity = eye(5);
% Gives a 5x5 identity matrix.

elements = [1, 2, 3, 4];
diagonal = diag(elements);
% You can also specify the elements on the diagonal with diag()
```

There are still more functions for creating variables for more niche cases, but these are the most common functions and operators for making vectors nd matrices.

Sizes/Dimensions

Because we have to deal with multidimensional arrays for basically most of programming in MATLAB, MathWorks has also provided some functions for retrieving information about the sizes and dimensions of our matrix variables.

- length() - Gives the length of the longest dimension
- size() - Gives the size of the array
- ndims() - Gives the number of dimensions of the array
- numel() - Gives the number of elements in the array

Reshaping

Sometimes, it is useful to change the shape of an existing matrix to a different size. The function reshape() does exactly that. It is important to note that MATLAB is a column-major language, which means that MATLAB stores its 2D matrices going down each column from left to right instead of by each row from top to bottom in memory. So, when reshape() fills in the new reshaped matrix, it fills the new matrix going down each column from left to right with the elements ordered in the column-major order stored in memory.

```
example = 1:16;
reExample = reshape(example, [2, 8]);

%{
(By the way this is how you make multiline comments)
This reshapes the vector into a 2x8 matrices with the contents
[
 1, 3, 5, 7,  9, 11, 13, 15
 2, 4, 6, 8, 10, 12, 14, 16
]
%}

rereExample = reshape(reExample.', 4, []);
%{
You can also let MATLAB automatically calculate the dimension
   size by using [].
This should output
[
 1,  9, 2, 10
 3, 11, 4, 12
 5, 13, 6, 14
 7, 15, 8, 16
]
%}
```

It is also important to know that the new matrix must have the same number of elements as the first element. So, if I tried to reshape the example above into a 4x5 matrix, MATLAB would not like that very much.

Broadcasting

The idea of broadcasting is a little weird, but it also makes MATLAB quite powerful. Sometimes when you perform operations on matrices of certain sizes, MATLAB performs repmat() on the input matrices to make the inputs sizes compatible. For example,

```
a = [1, 2, 4];
b = [5; 6; 7; 8];

c = a + b;
% c should give [6, 7, 9; 7, 8, 10; 8, 9, 11; 9, 10, 12] as an
   output
```

Even though the sizes are 1x3 and 4x1, the third line effectively performs [a; a; a; a] + [b, b, b]. While broadcasting, you do have to be very aware of what the dimensions of any given variable as you can introduce some incredibly annoying bugs if done improperly.

A New Type of Matrix

On the topic of making matrices, we can also introduce a new datatype in MATLAB that will come in handy next few lessons. The logical datatype effectively acts as a boolean for matrices with its own types of operators. For a short example of logical arrays,

```
logic = logical(eye(4));
% creates a logical matrix

example = reshape(1:16, 4, []);
great = example > 9;

%{
great should equal to
[
    0, 0, 0, 1
    0, 0, 1, 1
    0, 0, 1, 1
    0, 0, 1, 1
]
as the 9 is broadcast to fit the size of example
%}
```

In order to handle logical matrices, there are quite a few operators specifically designed for logical outputs.

Operator	Purpose
& and &&	Logical AND, Logical AND with shortcutting
| and ||	Logical OR, Logical OR with shortcutting
∼	Logical NOT
xor()	Logical XOR
all()	Determines if all values true/non-zero
any()	Determines if any values true/non-zero

Table 1: Logical Operations

Note: The shortcutting operations only work on single logical values.

Operator	Purpose
= and ∼=	Equality, Inequality
< and <=	Less than, Less than or equal to
> and >=	Greater than, Greater than or equal to
isequal()	Array equality
isequaln()	Arrat equality, counting NaN as equal

Table 2: Logical Operations

The functions true() and false() also effectively act as ones() and zeros() for logical arrays.

Documentation

Most of the functions that you have looked up for your first homework can also be applied to matrices. However, it is still important to read the documentation for each function as the behavior for each function may vary. For example, the exp(), sin(), and floor() functions mostly act elementwise by default, but max() and sum() act on each row by default. There may also be many different types of input formats that give different results.